```
/*
 * PanaIR_rx.c version 2
 *
 * Author : EH   Created: 2017.01.04 changed to v.2 2018.11.22
 * Receive specific remote Panasonic TV IR bit patterns, consisting of
 * frames of 48 bits e.g.  16 bit address + 32 bit data
 * Pressing a key on the Panasonic remote, it sends two identical frames
 * lasting approx. 63 msec. and approx. 75 msec. in between the frames.
 * NOTE: The PWR button transmits 3 identical frames.
 * PB2, pin 7 is INT0 , the input for the Sharp GP1UX511QS IR receiver
 * PB4, pin 2 is the for LED light control
 *
 * Panasonic remote buttons being used: 3D, left arrow, right arrow
 * IR code actions:
 * Pressing 3D twice toggles the PWM to the LED control output
 * While the PWM signal is enabled, pressing left arrow will decrease the PWM duty cycle
 * While the PWM signal is enabled, pressing right arrow will increase the PWM duty cycle
 *
 * 2018.11.22 - rewrite the code to include a state machine (STM) for button sequences
 * 2018.12.20 - left and right arrow only work from preceeding 3D, left-arrow, right-arrow
 * 2019.01.05 - included an extra STM state to refine LED control
 */

# define F_CPU 1000000UL    // 1 MHz clock with DIV8 fuse enabled
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/eeprom.h>
#include <util/delay.h>

// variables used by IR ISR
volatile unsigned int NextBit;
volatile unsigned int RecdAddress;
volatile unsigned long RecdDataH;
volatile unsigned long RecdDataL;
volatile unsigned long newFrame;

volatile uint8_t newFrameFlag;

#define PanaAddress 0x4004          // Remote Panasonic TV IR address code
#define PWRbutton 0x0100BCBD        // Remote PWR button IR data code
#define but3Dbutton 0x0190ED7C      // Remote 3D button IR data code
#define leftButton 0x01007273       // Remote volume down button IR data code
#define rightButton 0x0100F2F3      // Remote volume down button IR data code

#define OCRfreq 152         // OCR1B PWM upper count , approx. 100 Hz

#define INT0_PIN PORTB2     // IR receiver active low output attiny pin 7
#define LIGHT PORTB4        // light signal active low output attiny pin 2
#define REDLED PORTB0       // red led active low output attiny pin 5

int main(void)
{
    volatile uint8_t state, OCR1Bsave;

    DDRB &= _BV(INT0_PIN);      // set the INT0_PIN pin as an input
    PORTB |= _BV(INT0_PIN);     // pull up resistor on INT0_PIN input

    DDRB |= _BV(LIGHT);         // set the LIGHT pin as an output
    DDRB |= _BV(REDLED);        // set the REDLED pin as an output
    PORTB |= _BV(LIGHT);        // turn off the light , active low !
    PORTB |= _BV(REDLED);       // turn off the red led , active low !

    TCCR0A = 0;
    TCCR0B = 3<<CS00;           // T0 Prescaler /64 , 64 usec. timer
    GTCCR = 1<<PWM1B | 2<<COM1B0;
    TCCR1 = 7<<CS10;            // set Timer1 for PWM at PB4
    OCR1Bsave = OCRfreq / 2;    // start up duty cycle (50%)
    OCR1B = OCRfreq;            // no PWM at start
    OCR1C = OCRfreq;

    MCUCR |= (1<<ISC01);        // Interrupt on falling edge
    GIMSK |= _BV(INT0);         // Enable external interrupt INT0

    NextBit = 48;
    newFrameFlag = 0;           // clear new frame indicator flag
    state = 0;                  // clear state for the command state-event handler

    sei();                      // enable global interrupts

    while (1)
    {
     if (newFrameFlag == 1)         // got a new frame
     {
        cli();                      // disable frame interrupt while decoding frame data
        if (RecdAddress == PanaAddress)  //  frame with the Panasonic TV address ?
        {
            switch (state)
            {
                case 0: if (newFrame == but3Dbutton) state = 1; break;
                case 1:
                 if (newFrame == but3Dbutton)
                 {
                    if (OCR1B == OCRfreq)
                    {
                        OCR1B = OCR1Bsave;      // enable PWM with previous setting
                        state = 2;
                    }
                    else
                    {
                        OCR1Bsave = OCR1B;      // save PWM setting
                        OCR1B = OCRfreq;        // disable PWM
                        state = 0;
                    }
                 }
                 else state = 0;
                break;
                case 2:                 // Light is ON ( from state 1), check for PWM control
                    switch (newFrame)
                    {
                        case rightButton: if (OCR1B > 15) OCR1B = OCR1B - 10; break;
                        case leftButton:  if (OCR1B < OCRfreq - 20) OCR1B = OCR1B + 10; break;
                        case but3Dbutton: state = 3; break;
                        default: state = 0;
                    }
                break;
                case 3:
                 if (newFrame == but3Dbutton)
                 {
                    OCR1Bsave = OCR1B;      // save PWM setting
                    OCR1B = OCRfreq;        // disable PWM
                    state = 0;
                 }
                 else state = 0;
                break;
            }
        }
        _delay_ms(150);                 // delay for one interframe break plus one frame
        newFrameFlag = 0;               // clear newFrame flag
        sei();                          // enable interrupt for next IR frame
     }
    }
}

// Interrupt service routine, invoked on every falling edge of PB2, e.g. start of an IR pulse
// Timer0 intervals of 64 usec are used for measuring IR pulse width

ISR(INT0_vect) {
  int BitTime = TCNT0;
  int Overflow = TIFR & 1<<TOV0;
  if (NextBit == 48)    // looking for the Panasonic header period
  {                     // in between 4700 and 5700 usec.
    if ((BitTime >= 73) && (BitTime <= 89) && (Overflow == 0))
    {
      RecdAddress = 0;
      RecdDataH = 0;
      RecdDataL = 0;
      NextBit = 0;
    }   // got header, now ready to receive 48 bit data
  }
  else
  {
    if ((BitTime > 30) || (Overflow != 0))
       NextBit = 48; // max bit period exceeded, restart
    else
    {
      if (BitTime > 15) // if bit time > "0"-time, e.g. add "1" into the bit position
      {
          if (NextBit <= 15) RecdAddress = RecdAddress | ((unsigned int) 1<<(15-NextBit));
          else
          {
            if ((NextBit <= 31) && (NextBit > 15)) RecdDataH = RecdDataH | ((unsigned int) 1<<(
15-(NextBit-16)));
            else RecdDataL = RecdDataL | ((unsigned int) 1<<(15-(NextBit-32)));
          }
      }
      NextBit++;
      if (NextBit == 48)
      {
        newFrame = (RecdDataH << 16) | (RecdDataL & 0x0000FFFF);
        newFrameFlag = 1;
      }
    }
  }
  TCNT0 = 0;            // Clear counter
  TIFR |= 1<<TOV0;      // Clear overflow
  GIFR |= 1<<INTF0;     // Clear INT0 flag
}
```